Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 108C (2017) 2190–2199

Procedia Computer Science

International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland

Design Evaluation of a Performance Analysis Trace Repository

Richard Grunzke[1]*, Maximilian Neumann[1], Thomas Ilsche[1], Volker Hartmann[2], Thomas Jejkal[2], Rainer Stotzka[2], Andreas Knüpfer[1], and Wolfgang E. Nagel[1]

[1] Center for Information Services and High Performance Computing
Technische Universität Dresden, Dresden, Germany
richard.grunzke@tu-dresden.de
[2] Institute for Data Processing and Electronics
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

Parallel and high performance computing experts are obsessed with performance and scalability. Performance analysis and tuning are important and complex but there are a number of software tools to support this. One methodology is the detailed recording of parallel runtime behavior in event traces and their subsequent analysis. This regularly produces very large data sets with their own challenges for handling and data management. This paper evaluates the utilization of the MASi research data management service as a trace repository to store, manage, and find traces in an efficient and usable way. First, we give an introduction to trace technologies in general, metadata in OTF2 traces specifically, and the MASi research data management service. Then, the trace repository is described with its potential for both performance analysts and parallel tool developers, followed with how we implemented it using existing metadata and how it can utilized. Finally, we give an outlook on how we plan to put the repository into productive use for the benefit of researchers using traces.

Peer-review under responsibility of the scientific committee of the International Conference on Computational Science

Keywords: Performance Analysis, Data Repository, Metadata Extraction

1 Introduction

Many fields of science and research today are driven by data. The amount of data, the numbers of files, and the complexity of research data are growing steadily. Furthermore, data is not assigned to individuals anymore but teams are required to work with it in a joint manner. Due to this, organized data and metadata management is indispensable. It is the basis for collaborative work with data sets and it provides valuable benefits like 1) automatic annotation of advanced metadata; 2) advanced search capabilities for finding pieces of data; 3) interoperable (high performance) data access and sharing; 4) integration of standard data (pre-)processing; and 5) support for data preservation/archiving.

1877-0509
Peer-review under responsibility of the scientific committee of the International Conference on Computational Science
10.1016/j.procs.2017.05.190

In this paper we consider the scenario of trace-based performance analysis for parallel high performance computing. This method is based on a very detailed recording of the parallel execution of a target application which is then subject to post-mortem analysis. The findings of this analysis then guide the performance tuning process. Depending on various factors the involved trace files easily reach into the 3-digit gigabyte range for one execution of a parallel application. Usually, several traces need to be recorded in the course of one tuning project. Furthermore, the history from many such tuning projects is a valuable resource to learn from for future projects. Thus, this paper presents a very data intensive scenario with its own challenges for data and metadata management. We implemented the first version of a repository that is able to manage event trace files in a structured way. Researchers can search for traces according to various characteristics. This is a substantial shift, as before the traces were stored in a shared directory structure and users had to laboriously search for traces that meet their requirements in a manual way. Now, users can graphically search for the specific characteristics of traces based on structured metadata.

First, Section 2 describes the relevant trace technologies and the MASi service. Then, Section 3 details the use case, the implementation, and its usage. The move towards the sustainable operation is outlined in Section 4 while in Section 5 related work is presented.

2 Background

2.1 Trace Technologies

Performance analysis techniques can be characterized by a number of aspects. In this work, we look in particular at post-mortem performance analysis, where measurement data is collected during the experiment and written to persistent storage for later analysis. Further, we focus on tracing tools that log each individual measurement event during data recording, as opposed to profiling tools that summarize the collected data. Tracing results in a large amount of data, therefore it is particularly important to use metadata to improve the accessibility of performance measurements. A more detailed description of the classification of performance measurement tools can be found in [7].

There are several tools for performance analysis of HPC applications, most of them using their own measurement format. The Tuning and Analysis Utilities (TAU) [14] focus mostly on profiling, but also provide some support for tracing. HPCToolkit [15] measures the application using sampling and can record either profiles or traces. The Open Trace Format (OTF) [10] presents an effort towards a trace format that can be used independently of a specific tool. It was mainly developed as an output format for the VampirTrace [13] measurement tool. Similarly, its successor OTF2 [3] is associated with the Score-P measurement infrastructure [12]. Score-P and the resulting OTF2 measurements are used by the analysis and visualization tools Scalasca [4] and Vampir [13]. These tool chains can tackle parallel traces of up to several terabytes.

The data of a performance trace consists of a series of time-stamped events or samples that can have many different attributes. This variety of events in the trace data, already presents a complex opportunity for filtering and searching multiple traces. In addition, a wide range of metadata can be associated with a trace.

2.2 Metadata in OTF2 Traces

OTF2 traces created by Score-P contain various metadata:

- Most of the **definitions** in traces are necessary for reading trace events. Others describe the execution conditions. Typical definitions are:
  - A system tree, hierarchically organizing the parallel hardware from an entire parallel compute cluster over single compute nodes down to CPUs.
  - Paradigms used in the trace, e.g. `OPENMP`, `MPI` with specific properties.
  - Recorded locations and their type (e.g. `CPU_THREAD`),
  - Recorded regions (e.g. recorded functions).
  - Length of the trace and resolution of the timer.
- The **anchor file** (`*.otf2`) organizes the trace data set comprised of multiple files and contains as metadata for example:
  - OTF2 version and a unique trace identifier.
  - Number of locations, global definitions, snapshots, thumbnails, and properties.
  - Semantic properties of the trace, e.g. the completeness of recorded aspects.
- `scorep.cfg` is a separate file in the experiment directory, that contains all Score-P related environment variables utilized during the creation of the trace.

2.3 MASi Research Data Management Service

In the MASi DFG project [5] a research data management service is being build up based on the KIT Data Manager (KIT DM) framework [1, 8]. It is an open source framework for research data repository systems that can be largely adapted to many use cases. It enables the advanced management of research data using metadata while providing capabilities for data sharing, user and group management, automatic metadata extraction and data processing, and flexible file transfers. Besides a commandline client Java and RESTful APIs are provided while a web-based graphical user interface is currently developed (see below). The MASi project includes three community use cases: historical maps, spectroscopy in chemistry, and church windows [5].

A major aspect of MASi is to extend the KIT DM with various features. One is a generic REST API that abstracts from underlying metadata storages such as ElasticSearch or graph databases in order to enable seamless yet secure access to these. Another one is the implementation of a generic web-based user interface and web portal. Further new features include an OAI-PMH [16] (The Open Archives Initiative Protocol for Metadata Harvesting) module to enable the easy sharing of Open Access data. All extension have been or will be included as open source in the standard KIT DM version.

Another major aspect is to first incorporate the three use cases into the MASi service along the respective requirements. This first involves implementing automatic metadata extraction capabilities for XML and XMP as well as a implementing a graphical client-side GUI to enter metadata manually that can not be automatically extracted. And secondly to bring the MASi service into full production to be sustainably operated at the Centre for Information Services and High Performance Computing in Dresden, Germany.

3 Related Work

Bringing the two topics of data intensive parallel performance analysis and research data management together, there is little related work focusing on the topic presented here. All per-

formance analysis tools already collect some metadata as a basic prerequisite, compare 2.2 on page 2, and thus have their implicit or explicit metadata models.

As far as software tools for trace analysis are concerned, some support a comparative analysis [17], which means not only looking at one event trace at a time but at two or few. None employs an actual data management solution. Among the tools that rely on profiling instead of trace recording the TAU toolkit provides the *Performance Data Management Framework* (PerfDMF) which is essentially an SQL database ingesting the complete parallel profile data from many parallel runs [6, 9]. An explicit data and metadata model was created to which different formats need to be mapped. To the best of our knowledge a performance evaluation is not published for large-scale parallel traces. However, storing large event trace data sets in SQL databases seems infeasible as it would not allow storage interactive work with multi Gigabyte to Terabyte event traces. Also, the approach necessitates to implement a mapping from every format to be ingested to the unified model and the access and sharing appears to be limited to the local systems. In contrast, our approach is more flexible as it only requires a basic metadata extraction method to be implemented (see Section 4.2). This extraction enables the inclusion of the metadata into the integrated NoSQL index of the ElasticSearch search software that provides highly scalable and advanced search functionality. Furthermore, our approach lends itself to directly publishing traces to a wide audience via the MASi service. The focus of PerfDMF is on profiles which are generally smaller. This contrasts with the focus of our repository which handles traces that are generally larger.

In the field of research data management there are no published results focusing on trace data. With the KIT DM framework and the MASi service, we are utilizing a solution that is designed for both large-scale data management and to be flexibly adapted to specific use cases such as the event trace data at hand.

The next section describes how the trace repository was build within the MASi service.

4 A Trace Repository

4.1 The Performance Analysis Use Case

Parallel performance analysis using event traces is a data-intensive occupation as mentioned in the introduction. Still, it is not limited to looking at one trace data set ("a trace" in short) at a time. Instead, working with many traces is essential in at least two ways.

The Performance Analyst's Perspective

Even though the typical work of a parallel performance analyst focuses on one application code at a time, this still involves many traces.

First, there is the regular feature of various tools for parallel performance analysis to compare two or more traces in detail. Examples are Vampir [11] and TAU [14]. Typical questions for comparative analysis of traces for a given program code are

- Compare the **same code** executed on **different parallel machines**.
- Compare the **same code** executed with **different inputs or parameters**.
- Compare the **same code** at **different parallel scales**.
- Compare the **same code** with **different/hybrid parallelization models**.
- Compare the **same code** on the **same machine** with the **same settings** to study **variability, reproducibility, or non-deterministic effects**.

Second, comparative performance analysis is interested in changing behaviour respectively performance during the course of the optimization. That means, how does the achieved performance evolves for successive code versions implementing different (potential) code optimizations. Eventually, this also answers the question, how the finally achieved performance compares to the starting point of the optimization endeavour.

Third, an overview over many/all traces from past optimization projects might be very useful when starting the next optimization project. If one could easily identify past cases similar to the current one, this might give valuable hints such as:

- What was the final optimized performance? This might allow a better estimate about the optimization potential because it relates to the realistic performance level of a real application code instead of the machines theoretical peak performance.
- What were successful or unsuccessful optimization strategies? Then one would apply the previously successful ones first.

Albeit there is probably no good general definition of "similar code" or "similar trace", it is applicable if focusing on one aspect at a time, for example "similar ratio of computation to parallel communication", "similar in terms of floating point operations vs. memory accesses", or "similar in I/O behavior". Then different traces will be used as references for different performance aspects. From this, one can infer a number of useful metadata categories for a trace data repository, that are valuable for a performance analyst:

Configuration: code version, configure and compiler flags, linked libraries, compiler versions
Inputs: run time parameters and descriptive names or just hashes of input data
Parallelism: level of parallelism, broken down for hybrid parallelism if applicable
Platform: characterization of the compute nodes and the network topology

From each category follows a list of individual metadata fields.

The Perspective of the Tools Developer

How does the perspective of a software tool developer for parallel performance analysis compares to this? His or her main concern is not a parallel code under analysis but the tools for analysing and visualizing traces that require a repertoire of input traces. Primarily, this is for testing and validation. Testing requires a broad spectrum of input traces to check whether the tools can properly process them. Extreme cases w.r.t. certain characteristics are especially interesting, for example maximum total size, maximum number of processes/threads, largest degree of imbalance in the amounts of data per process/thread. Validation needs a set of input traces together with reliable performance results in order to check if the tools produce the same result. Examples are the total number of recorded trace events, the number of subroutine calls or the maximum value of a performance metric. As a secondary concern, tools developers want to be able to search for traces with special characteristics so that they find test cases for a new type of analysis. Also, searching for suitable demonstration cases would be useful, that include given properties or combinations of properties. From this perspective, in addition to all the metadata categories from the previous list the following ones are required:

Characteristics of the code. Which parallel paradigms used, which libraries, ...?
Characteristics of what was recorded in the trace. Which subroutines, library calls, hardware performance counters, processes/threads, ... have been covered or excluded. [1]

[1] The monitoring scope needs to be limited, keeping all aspects would surge data volumes and overheads.

Selected performance analysis results. What are dominating activities (like computation, communication, or synchronization phases)? What are dominating runtime symptoms (like load imbalances, heavy I/O, floating point operations vs. integer operations, ...)? What was the estimated monitoring overhead (distortion)?

Reproducible checksums and conditions. Known analysis results that should be exactly reproduced when analyzing one trace anew.

The Current Local Data Management Scheme

The current trace data management scheme used by a mixed group of parallel performance analysts and developers of parallel performance analysis tools consists of a shared data container. Currently it contains 3229 trace data sets, which usually comprise of multiple files each. In total it contains 17 TB, where the largest individual trace is 748 GB alone.

Structuring is currently done via a directory tree, where the top level are personal directories for the creators of the trace data sets, that are past and present colleagues as well as external partners. The following one or more levels down the directory tree are subdirectories with more or less expressive names. Every creator is free to pick his or her own substructuring, though. In few cases (less than 5%) there is a readme file next to the trace data set.

Finding a particular trace from a different creator in this structure is complicated. Adding a newly created trace to this data container (ingest) is the exception and not the norm. That is because the volume of the data container is limited, everybody knows that arbitrary additions will exceed a sensible volume soon, even though there is no strict limit from the hosting computing center. Only traces that someone considers to be extraordinarily interesting while working with it are added. If one comes to the conclusion later that one case would have been interesting to keep, it is usually gone. There is no regular or somewhat organized removal procedure for that data container.

One advantage of this solution is the close proximity to the supercomputer, where most new traces are generated and where even the biggest traces can be processed with parallel tools. Even though no structured metadata is kept next to the trace data sets, they do contain some metadata (see Section 2.1). However, this is not easily accessible via search queries.

Required Functionality

For the newly designed trace data repository, the following requirements and issues need to be addressed:

Searching A quick and efficient searching mechanism for traces is essential. It should show all criteria and support filters. For this, we also plan to gather information that is not included in trace files themselves but are highly useful nonetheless (compiler name and version, system characteristics (CPU, RAM, ...), utilized library versions, ...).

Exploration The next step after searching is exploration of the available data sets. That means step by step refining search criteria, thereby adding or modifying filter criteria until a satisfying subset of results is reached.

Automated ingest There should be an easy ingest procedure to bring even huge traces into the repository while at the same time supporting the automatic metadata extraction.

Fast data access While utilizing the trace repository, the full trace data sets must still be accessible from the local supercomputer with high bandwidth.

Access management The repository should allow external access to the trace data sets, either publicly (Open Access) or for selected individuals. By default trace data sets are only accessible for their creator. Also important is to select an appropriate license.

Informed data removal Removing traces from the repository is inevitable because of the sheer data volumes. The decision which ones to remove needs well-informed.

They serve as guidelines for the following design and implementation.

4.2 Implementation and Usage

The trace repository was implemented within the MASi research data management service 2.3 and is graphically accessible as a subsection within the MASi portal. The following paragraphs describe how metadata is extracted, how the traces are ingested, and how traces can be graphically searched for and downloaded.

Metadata Extraction

The OTF2 metadata extractor is a parallel program based on the OTF2 library. It reads either only definition records, i.e. metadata records describing the trace, which always make up small amounts of data. In this case the I/O and computing times are negligible. Or it scans the entire parallel trace which consists of multiple large data files (usually one per original process/thread). Those files are processed in parallel in a 1:n fashion, i.e. each extractor process reads multiple data files. Each extraction process needs to read the entire data files but only processes the data records of interest, e.g. reading subroutine call events but ignoring hardware performance counter samples. The essential information collected are:

- Path: the path to anchor file of a multi-file trace
- NumLocations: number of recorded processes/threads
- Paradigms: used parallel programming models (e.g. COMPILER, MPI)
- System tree: hardware configuration of the parallel execution environment
- Function statistics: summary of function calls including function names, call counts, and exclusive and inclusive executions times (*)
- Metrics: Summary of recorded metrics, esp. hardware performance counter samples (*)

All but the last two (marked with *) are extracted from trace definitions. Only the function statistics and metrics require reading the traces completely. The extracted metadata are stored in XML form, see Listing 1.

Listing 1: Excerpt from an example XML file with metadata of a trace

```
<?xml version="1.0"?>
<metadata xmlns="http://tu-dresden.de/zih/tracerepository">
<File>/trcdata/tracefiles/tschueter/bt-mz_C.128x2_trace_instrumented/
    traces.otf2</File>
<NumLocations>256</NumLocations>
<Paradigm index="0">COMPILER</Paradigm>
<Paradigm index="1">MPI</Paradigm>
<Paradigm index="2">OPENMP</Paradigm>
<Paradigm index="3">USER</Paradigm>
<node index="0">machine taurus.hrsk.tu-dresden.de
<node index="1">node taurusi1044
<node index="2">MPI Rank 0
<node index="3">Master thread:0</node>
```

Figure 1 shows a first evaluation of the extraction speed. In this example the full metadata set was extracted from traces in OTF2 format with sizes from 230 MiB to 169 GiB. All traces were processed with the extraction tools on an exclusively used dual socket Intel "Haswell" compute node (dual Intel Xeon E5-2680 v3, 2x12 cores, 2.50GHz, multi threading disabled, 64 GB Ram). When the traces are processed directly from their normal storage location in an NFS container, the speed is constant on a level of approx. 110 MB/s (shown in blue). When the data is copied to a high-performance Lustre parallel filesystem this rises to 800 to 1600 MB/s for traces that are not too small. This shows that the extraction is I/O bound and not compute bound.

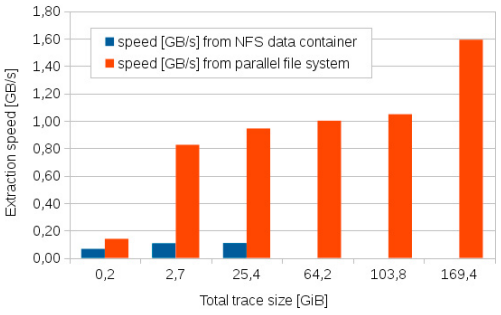

Figure 1: Evaluation of the extraction performance.

Currently, the command line based metadata extraction method is the lowest common denominator to support any data format. Meaning that when others formats (see Section 2.1 besides OTF2 shall be supported, further such extraction methods needs to be implemented.

Trace Data Ingest, Search, and Download

The command line client of the KIT DM (repoClient) it utilized in order to ingest traces. To facilitate an seamless integration on the supercomputer Taurus [2], the repoClient was installed via the module environment. Besides the command for loading the module (line 1), Listing 2 shows the command that initializes the environment of the user with the necessary information such as login/password, active group, and service URL (line 2). The command in line 3 in Listing 2 shows an example to ingest a trace into the MASi repository. The trace is then uploaded utilizing the WebDAV protocol together with the XML file containing the metadata (see 1). Post-uploading, the metadata is automatically extracted from the XML file and incorporated into the ElasticSearch search index of the MASi service. Figure 2 shows the graphical search interface of the trace repository. It currently allows for searching via complex queries, browsing through results and downloading them for further use. As seen in Figure 2, they can be download via the a web browser. Alternatively, the wget-command (see Listing 2 line 4 and 5) can be used to directly download a trace on an HPC system.

Listing 2: Loading the module, initializing the environment, ingesting and downloading a trace.

```
module load repoclient                                                    1
repoClient init                                                           2
repoClient ingest -n "Trace" -i bt-mz_C.128x2_trace_instrumented/         3
wget 'http://masi.zih.tu-dresden.de:9090/KITDM/rest/dataorganization/     4
organization/download/187/?groupId=CVMA&viewName=default&authToken=...'   5
```

5 Conclusion and Outlook

Most modern research data management systems, employing metadata, are specifically designed and implemented for single use cases or a narrow range of them. They can not with limited effort be adapted to other use cases. To significantly mitigate this signifies a novelty of the

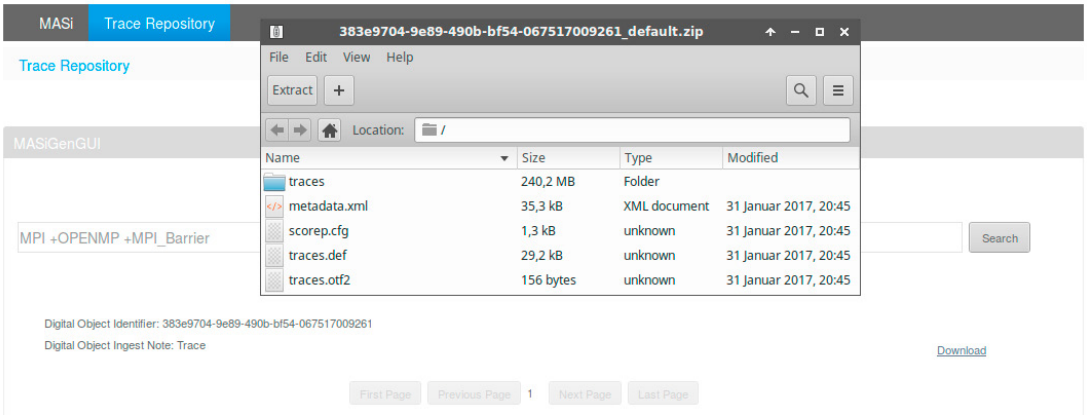

Figure 2: This figure shows the graphical search interface with a search query, the result below and a downloaded trace.

KIT DM framework and the MASi service. They can be adapted in a limited amount of time to new and heterogeneous use cases while providing advanced features. Depending on various factors, now only weeks are required to create community specific repository instead of years, as before. Exploiting these characteristics, we designed a trace repository able to handle large, heterogeneous, and complex trace files with the potential to vastly improve the work of performance analysts. Within the MASi research data management service we implemented a functional version, showed its potential and gave an outlook on how to bring it into production. With this, teams working with many very large trace data sets become able to organize them in structured and collaborative manner and identify interesting ones while avoiding the need to scan entirely again and again.

In order to transition the trace repository from a first functional version into production, two main steps are planned. First, independent of the trace repository, the MASi service is currently being advanced to be offered as an official computing center service to a wide range of users and with long-term availability and maintenance. Second, we plan to implement the features detailed in Section 4.1 to provide advanced capabilities to satisfy the requirements of users utilizing trace data.

Acknowledgments

This work was supported by the German Research Foundation via the project MASi and the Collaborative Research Center 912 HAEC.

References

[1] KIT Data Manager. `http://datamanager.kit.edu/`, December 2016.

[2] Taurus Supercomputer at ZIH. `https://tu-dresden.de/zih/hochleistungsrechnen/hpc`, 2017.

[3] Dominic Eschweiler, Michael Wagner, Markus Geimer, Andreas Knüpfer, Wolfgang E. Nagel, and Felix Wolf. Open Trace Format 2: The Next Generation of Scalable Trace Formats and Support Libraries. In *Applications, Tools and Techniques on the Road to Exascale Computing*, volume 22 of *Advances in Parallel Computing*, pages 481 – 490, 2012.

[4] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Ábrahám, Daniel Becker, and Bernd Mohr. The Scalasca Performance Toolset Architecture. *Concurrency and Computation: Practice and Experience*, 22(6), April 2010.

[5] Richard Grunzke, Volker Hartmann, Thomas Jejkal, Ajinkya Prabhune, Sonja Herres-Pawlis, Alexander Hoffmann, Aline Deicke, Torsten Schrade, Hendrik Herold, Gotthard Meinel, Rainer Stotzka, and Wolfgang E. Nagel. Towards a Metadata-driven Multi-community Research Data Management Service. In *2016 8th International Workshop on Science Gateways (IWSG)*, 2016, accepted.

[6] Kevin A Huck, Allen D Malony, Robert Bell, and Alan Morris. Design and implementation of a parallel performance data management framework. In *Parallel Processing, 2005. ICPP 2005. International Conference on*, pages 473–482. IEEE, 2005.

[7] Thomas Ilsche, Joseph Schuchart, Robert Schöne, and Daniel Hackenberg. Combining Instrumentation and Sampling for Trace-Based Application Performance Analysis. In *Tools for High Performance Computing 2014*, pages 123–136. Springer International Publishing, 2015.

[8] Thomas Jejkal, Alexander Vondrous, Andreas Kopmann, Rainer Stotzka, and Volker Hartmann. KIT Data Manager: The Repository Architecture Enabling Cross-Disciplinary Research. *Large-Scale Data Management and Analysis (LSDMA) - Big Data in Science*, pages 9–11, 2014.

[9] Karen L Karavanic, John May, Kathryn Mohror, Brian Miller, Kevin Huck, Rashawn Knapp, and Brian Pugh. Integrating Database Technology with comparison-based parallel Performance Diagnosis: The Perftrack Performance Experiment Management Tool. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 39. IEEE Computer Society, 2005.

[10] Andreas Knüpfer, Ronny Brendel, Holger Brunst, Hartmut Mix, and Wolfgang E. Nagel. Introducing the Open Trace Format (OTF). In *6th International Conference on Computational Science (ICCS)*, volume 2, pages 526–533, Reading, UK, 2006. Springer.

[11] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S Müller, and Wolfgang E Nagel. The Vampir Performance Analysis Tool-set. In *Tools for High Performance Computing*, pages 139–155. Springer Berlin Heidelberg, 2008.

[12] Andreas Knüpfer, Christian Rössel, Dieteran Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011*, pages 79–91. Springer Berlin Heidelberg, 2012.

[13] Matthias S. Müller, Andreas Knüpfer, Matthias Jurenz, Matthias Lieber, Holger Brunst, Hartmut Mix, and Wolfgang E. Nagel. Developing Scalable Applications with Vampir, VampirServer and VampirTrace. In *Parallel Computing: Architectures, Algorithms and Applications*, volume 15 of *Advances in Parallel Computing*, pages 637–644. IOS Press, 2008.

[14] Sameer S. Shende and Allen D. Malony. The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[15] Nathan R Tallent, John Mellor-Crummey, Michael Franco, Reed Landrum, and Laksono Adhianto. Scalable fine-grained Call Path Tracing. In *Proceedings of the international conference on Supercomputing*. ACM, 2011.

[16] Herbert Van de Sompel, Michael L Nelson, Carl Lagoze, and Simeon Warner. Resource Harvesting within the OAI-PMH Framework. *D-lib magazine*, 10(12):1082–9873, 2004.

[17] Matthias Weber, Ronny Brendel, and Holger Brunst. Trace File Comparison with a Hierarchical Sequence Alignment Algorithm. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 247–254. IEEE, 2012.